Poisoning the Kad Network*

Thomas Locher[1], David Mysicka[1], Stefan Schmid[2], and Roger Wattenhofer[1]

[1] Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Zurich, Switzerland
{lochert,wattenhofer}@tik.ee.ethz.ch, dmysicka@ethz.ch
[2] Deutsche Telekom Laboratories, TU Berlin, Berlin, Germany
stefan@net.t-labs.tu-berlin.de

Abstract. Since the demise of the Overnet network, the Kad network has become not only the most popular but also the only widely used peer-to-peer system based on a distributed hash table. It is likely that its user base will continue to grow in numbers over the next few years as, unlike the eDonkey network, it does not depend on central servers, which increases scalability and reliability. Moreover, the Kad network is more efficient than unstructured systems such as Gnutella. However, we show that today's Kad network can be attacked in several ways by carrying out several (well-known) attacks on the Kad network. The presented attacks could be used either to hamper the correct functioning of the network itself, to censor contents, or to harm other entities in the Internet not participating in the Kad network such as ordinary web servers. While there are simple heuristics to reduce the impact of some of the attacks, we believe that the presented attacks cannot be thwarted easily in any fully decentralized peer-to-peer system without some kind of a centralized certification and verification authority.

1 Introduction

Peer-to-peer (p2p) computing is one of the most intriguing new networking paradigms of the last decade. Not only do structured p2p systems, which typically implement a distributed hash table (DHT), possess crucial advantages over centralized systems for applications such as reliable data dissemination, structured p2p systems may also play a pivotal role in the challenging endeavor to redesign the Internet due to their valued properties such as small routing tables, fault tolerance, and scalability.

In this paper, we question whether the p2p approach is mature enough to step outside of its "comfort zone" of file sharing and related applications. In particular, not much is known about the ability of DHTs to meet critical security requirements (as those required nowadays, e.g., for domain name servers) and its ability to withstand attacks. To this end, as a case study, we evaluate the feasibility of various attacks in the Kad network, as it is currently the most widely deployed p2p system based on a DHT with more than a million simultaneous users [15].

Our study reveals that while the Kad network functions reliably under normal operation, today's Kad network has several critical vulnerabilities, despite ongoing efforts on

* Research in part supported by the Swiss National Science Foundation (SNF).

the developers' part to prevent fraudulent and destructive use. This paper describes several protocol exploits that prevent peers from accessing particular files in the system. In order to obstruct access to specific files, file requests can be hijacked, and subsequently, arbitrary information can be returned instead of the actual data. Alternatively, we show that publishing peers can be overwhelmed with bogus information such that pointers to the original files can no longer be accessed. Moreover, it is even possible to *eclipse* certain peers, i.e., to fill up their routing tables with information about malicious peers, which can subsequently intercept all messages. Additionally, we briefly discuss how our network poisoning attacks can also be used to harm machines outside the Kad network, e.g. web servers, by tricking the peers into performing a Kad-steered distributed denial of service (DDoS) attack. It is virtually impossible to determine the true culprit in this scenario, as the peer initiating the attack does not take part in the attack, which makes this kind of attack appealing to malicious peers.

All our attacks have been tested on the real Kad network using a modified C++ eMule client. Already with three attackers, virtually no peer in the system was able to find content associated with any given keyword for several hours, which demonstrates that with moderate computational resources access to any targeted content can be undermined easily.

The remainder of this paper is organized as follows. The basic functionality of Kad is described in Section 2. We present and evaluate our attacks in Section 3. Section 4 briefly discusses crucial implications of the attacks and points out the difficulty of finding effective countermeasures. After reviewing related literature in Section 5, the paper concludes in Section 6.

2 eMule and Kad

The Kad network is a DHT-based p2p network that implements the *Kademlia protocol* [10]. Access to the Kad network is provided through the *eMule*[1] client, which can also connect to the server-based *eDonkey*[2] network.

Each peer in the Kad network has a 128-bit identifier (ID) which is normally created by a random generator. This ID is stored at the peer even after it has left the network and is re-used once the peer returns. Routing in the network is performed using these identifiers and the XOR metric, which defines the distance between two identifiers as the bitwise exclusive or (XOR) of these identifiers interpreted as an integer. For all $i \in [0, 127]$, every peer stores the addresses of a few other peers whose distance to its own ID is between 2^i and 2^{i+1}, resulting in a connected network whose diameter is logarithmically bounded in the number of peers. For each of these *contacts* in the routing table, a Kad ID, an IP address, and a port is stored.

The publish and retrieval mechanisms work roughly as follows. Each keyword, i.e., a word in a file name, and the file itself, are hashed, and information about the keywords, its associated file, and the address of the owner is published in the network, i.e., this information is stored at the peers in the DHT whose identifers are closest to the respective hash values. More specifically, in Kad, information is replicated ten times in a

[1] See http://www.emule-project.net/

[2] See http://en.wikipedia.org/wiki/EDonkey_network/

zone where peers agree in the first 8 bits with the published key. Usually, this so-called *tolerance zone* contains several thousand peers. While most of the peers are very close to the key, this is not always the case, e.g., due to churn and also for keys that are very popular and published by many different peers.

If a peer wants to download a file with a certain name (a particular sequence of keywords), it first queries the peer whose identifier is closest to the hash of the *first* of the specified keywords, and this peer returns the information of all files whose file names contain all the given keywords, and also the corresponding file hashes. The requesting peer p_1 can then download the desired file by querying the peer p_2 whose identifier is closest to the file hash, as p_2 keeps track of all the peers in the network that actually own a copy of the file.

3 Attacks

This section presents three different attacks which limit the Kad network's participants access to a given file f. In a *node insertion attack*, an attacking peer seeks to attract search requests for f, which are answered with bogus information. Alternatively, access to f can be denied by filling up the index tables of other peers publishing information about f (*publish attack*). Finally, we describe how an attacker can *eclipse* an arbitrary peer: By controlling all the peer's incoming and outgoing traffic, the attacker can prevent a peer from either publishing information about f or from accessing it.

3.1 Node Insertion Attack

By performing a *node insertion attack*, it is possible to corrupt the network by spreading polluted information, e.g., about the list of sources, keywords, or comments. We have implemented the attacks for *keywords*, that is, a search for the attacked keyword will not give the correct results, but instead arbitrary data chosen by the attacker is returned.

For this attack to work, we have to ensure that the search requests for the specific keyword are routed to the attacking peer rather than to the peers storing the original information. This can be achieved as follows. In the Kad network, a peer normally creates its ID using a random number generator; however, any alternative mechanism will work as well, as there is no verification of a peer's ID. In our modified eMule client, it is possible to select the peer's Kad ID manually. Thus, an attacker can choose its ID such that it matches the hash value of the targeted keyword. Consequently, the peer will become the node closest to this ID and will receive most of the corresponding search requests. The nodes storing the correct files typically have a larger distance to the keyword's ID than the attacker, as the probability for a peer to have a random ID that perfectly matches the 128-bit keyword ID is negligible.

In order to guarantee that peers looking for a certain keyword only receive faked results, the attacker must provide enough result tuples, as the eMule client terminates the search after having gathered 300 tuples. The attacker further has to include the keywords received from a peer in the filenames, otherwise the replies are not accepted. In our attacks, we use filenames that contain a unique number, the message "File removed from Kad!", and the keywords. Unique file hashes are needed such that the 300 tuples are not displayed as one tuple in eMule's search window.

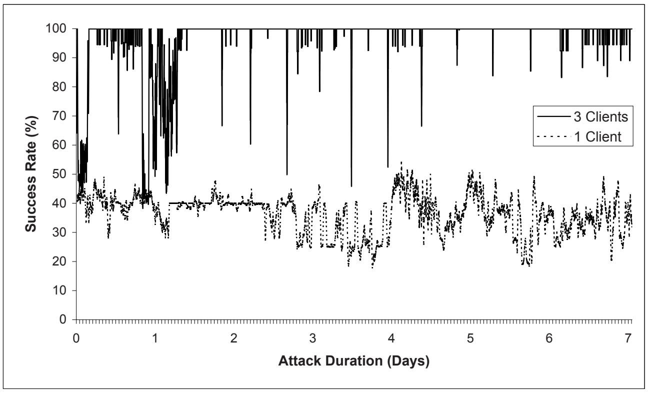

Fig. 1. Percentage of successfully hijacked keyword requests in a node insertion attack for 1 and 3 attackers during a time period of one week

We frequently observed that eMule sends search requests not only to the closest peer, even though this peer provided enough answers. This can be explained by the delay caused when transmitting the 300 search results from the closest peer. eMule will send another request to the second closest peer before all of the results are received from the closest one. This of course may harm the effectiveness of the attack, and hence it is beneficial to gain control over the second, third, etc. closest IDs as well by means of additional attackers. These attackers behave exactly the same way: All requests are answered by supplying 300 faked tuples.

Figure 1 depicts the traces obtained during two week-long node insertion attacks performed using our modified eMule client on the keyword "Simpsons." Note that this attack influences all queries in the entire Kad network not only for the search term "Simpsons", but also all other queries starting with the term "Simpsons" such as "Simpsons Movie" or "Simpsons Soundtrack" etc. are affected automatically.

In the first trace, only one attacker whose ID exactly matches the hash of the keyword infiltrated the network. We used another client to search for the term "Simpsons" once a minute and examined the returned results. Since a single attacker is not sufficient, as mentioned before, the attack is moderately successful in that only approximately 40% of the returned results originated from the attacker. What is more, every single query returned at least some results that are not faked. Further experiments showed that using two attackers instead of one does not increase the success rate substantially, but three attackers is already enough to hijack virtually all requests. The second trace shows the success rate of the node insertion attack using three attackers. On average, more than 95% of all returned tuples were faked, and every batch of tuples contained at least some bogus data created by the attackers. The plot shows that there are sudden drops of the success rate once in a while. An explanation for this behavior is that peers join and leave the network at a high rate, resulting in inaccurate routing tables. Consequently, a lookup request can be routed to a peer that still stores results for this request and does not know about our attacking peers yet.

The attack was repeated at other times using different keywords. All our other experiment resulted in a similar picture and confirmed our findings made with the "Simpsons" keyword. Our attacking peers received roughly 8 requests per minute from other peers in the network during the experiments. As expected, the peer having the closest ID received the most requests at a rate of roughly 4 requests per minute.

3.2 Publish Attack

In contrast to the node insertion attack, which forces the search requests to be routed to the attacker, the publish attack directly attacks the peers closest to the ID of the attacked keyword, comment, or source entry. The index tables stored by the peers in the Kad network have a limited length; for instance, the keyword table can store up to 50,000 entries for a specific ID. Moreover, a peer will never return more than 300 result tuples per request, giving priority to the latest additions to the index table. This makes it possible to replace the original information by filling up the tables of the corresponding peers with poisoned entries. Thus, an attacker seeks to publish a large amount of information on these peers. Once the index tables of the attacked peers are full, they will not accept any other publish requests by other peers anymore. Therefore, the attacked peers will only return our poisoned entries instead of the original information. Since every entry has an expiration time (24 hours for keyword and comment entries, and 5 hours for source entries), the clients have to be re-attacked periodically in order to achieve a constant fraction of poisoned entries. In addition, an attacker has to take into consideration the newly joining peers in the network; if they have an ID close to the one attacked, their tables also have to be filled.

We have implemented the publish attack for keyword entries as well, again by modifying the original eMule application. An existing timer method is used to run the attack every 10 minutes. In the first phase, the 12 peers closest to the targeted ID are located using eMule's search mechanism. In each run, only peers are selected that have not been attacked before or that need to be re-attacked due to the expiration of the poisoned entries. In the second phase, all the peers found in the first phase are attacked, beginning with the closest peer found. In order to guarantee a full cache list, 50,000 poisoned entries are sent divided into 250 packets containing 200 entries each. In order to prevent overloading the attacked client, the sending rate was limited to 5 packets per second. Every entry consists of a unique hash value and filename as in the node insertion attack. Since these entries ought to match all search requests containing the attacked keyword, it is necessary to include all additional relevant keywords (e.g. song titles for an interpreter, year and language for a film title) in the filename; otherwise, all the lookups with additional keywords would not receive the poisoned entries, because not all the keywords are included. In the node insertion attack, this problem does not occur as the additional keywords are obtained from every search request and can directly be appended to the filename to match the request. The success of each run is measured with the load value sent in every response to a publish packet. This value should increase with every poisoned packet sent, from a starting level of about 10 - 20% to 100% when the attack is finished.

In comparison to the node insertion attack, it is clearly harder to maintain a high success rate using the publish attack, due to the permanent arrivals of new peers and

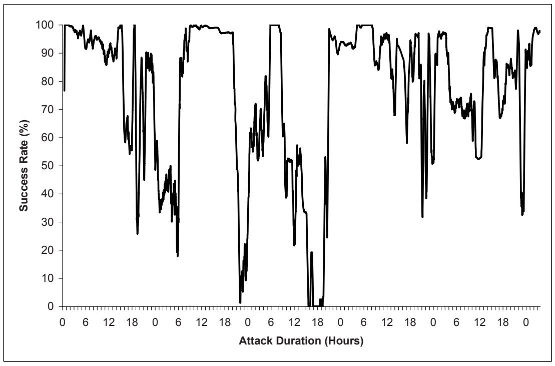

Fig. 2. Percentage of faked replies received in a publish attack for the keyword “Simpsons” during a time period of 5 days. Sometimes, the success rate drops but then recovers again quickly.

the need to re-attack the peers periodically. While the node insertion attack yields constantly high rates, this is not true for the publish attack. Figure 2 plots the success rate of an attack on the keyword “Simpsons” over a period of 5 days. While the attack works fairly well on average, at a success rate of roughly 80%, the success rate periodically drops and remains low for a certain time period before it recovers again.

Overall, the success rate is much lower than in the case of a node insertion attack, although performing a publish attack is much more expensive. Again, repeating the attack at other times using different keywords results in a similar pattern. The reason for this peculiar behavior is that the peers responsible for the targeted IDs that are online during the phase where the success rate is low refuse to accept our publish messages. In fact, these peers do not even reply to publish messages, even though they can be contacted, otherwise we could not receive any lookup results from them. As this behavior is not in accord with the protocol implemented in the real eMule client, we suspect that modified versions of the original client cause this effect. What clients are used is hard to determine as they do not directly provide this information. Thus, the use of modified clients appears to be another reason why the node insertion attack is superior to the publish attack. In order to improve the success rate, a more sophisticated implementation could be used where the attack is split up into two concurrent processes. The first one would permanently search for new peers with an ID close to the one attacked and pass them to the second process which would then attack these peers simultaneously. This would minimize the time during which peers can respond with original data. As this improvement would not solve the problem of uncooperative peers, it was not implemented.

3.3 Eclipse Attack

Instead of poisoning the network to keep peers from obtaining certain information, we can also attack the requesting peers directly and keep them from sending requests into the Kad network. In the eclipse attack, the attacker takes over the targeted peer’s routing table such that it is unable to communicate with any other peer in the Kad network except the attacker. As the attacker simulates the whole Kad network for that peer, it can manipulate the attacked peer in arbitrary ways, e.g., it can specify what results are returned for any lookup, or modify comments for any file. The peer's requests can also be directed back into the Kad network, but modified arbitrarily.

Typically, the contacts in the Kad routing table are not uniformly distributed over the whole ID space. Rather, most of the contacts are located around the peer's ID to maintain short lookup paths when searching for other peers in the Kad network (cf. [10]). The attacker takes advantage of the fact that there are relatively few contacts in most parts of the ID space. Concretely, we inject faked peer entries into these parts of the routing table to achieve a dominating position. Subsequently, the faked peers are selected for almost all requests. If we set the IP address of all those faked entries to the address of our attacking peer, we receive most requests of the attacked peer and can process them as desired. We make use of the fact that the standard eMule client accepts multiple neighbors of the same IP address.

Our measurements showed that a peer running eMule for an extended period of time has up to 900 contacts in its routing table. As the maximum number of contacts is 6,310, there is plenty of space in the routing table for faked entries. In order to inject faked entries the *Hello Request* message is used, which is normally utilized during connection set up to check whether known peers are still alive. As a side-effect of this message, the sender of the message is added to the receiver's routing table. After enough entries are injected, the attacking peer has to process the requests from all those entries in order to keep them in the routing table of the attacked node.

We implemented the eclipse attack in a stand-alone application and ported all necessary parts from the source code of eMule. The application maintains a list that holds all faked entries sent to the attacked peer. This is necessary, because every new entry in the routing table is validated by sending a hello request. This request has to be answered

Table 1. Percentage of faked replies received during 10 runs of the eclipse attack. Each run r was measured 15 minutes with an interval of one minute.

Minute	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	$\bar{r}$
1.	0	0	0	0	0	0	0	0	0	0	0
2.	0	0	0	0	0	0	0	0	0	0	0
3.	0	0	0	0	0	0	0	0	0	0	0
4.	0	0	0	81	0	0	78	0	0	0	15.9
5.	72	100	100	65	23	100	81	81	100	65	78.7
6.	78	100	90	72	85	100	78	72	100	81	85.6
7.	81	82	100	81	78	81	100	78	100	100	88.1
8.	65	100	100	100	81	100	100	68	81	100	89.5
9.	58	100	100	95	100	100	100	89	100	100	94.2
10.	78	100	100	100	100	100	98	100	100	100	97.6
11.	100	100	100	100	100	100	100	100	100	100	100
12.	100	100	100	100	100	100	100	100	100	100	100
13.	100	100	100	100	100	100	100	100	100	100	100
14.	100	100	100	100	100	100	100	100	100	100	100
15.	100	100	100	100	100	100	100	100	100	100	100

with the same ID as we have chosen when injecting the entry. In order to differentiate between the entries, we assign a new port to every faked entry and maintain a data structure to store this information. The other part of our application processes the requests of the attacked peer. If it asks for new peers close to a specific ID, we reply with new faked peers that match this ID, or are very close to it, to guarantee the success of the attack. If the peer asks for stored information we deliver poisoned results, as in the two attacks discussed before.

Table 1 shows the results of 10 repeated eclipse attacks under the same conditions. To measure the success rate of the attacks, we periodically ran searches on the attacked peer and counted the number of poisoned results. As the success rate virtually always reaches 100% within minutes, we can conclude that the attack works well, especially if the attack is focused on a single keyword, but it is naturally limited to merely a single attacked peer. The other two attacks are clearly preferable if an attacker aims at hiding content from *all* peers.

4 Discussion

The preceding section has presented three different attacks that can be used to keep peers from acquiring the requested information. Naturally, these attacks can also be combined in order to increase the chances of a successful attack. However, these poisoning attacks cannot only be used for this purpose. Rather, they can serve an attacker as basic building blocks to pursue completely different aims.

We will now briefly illustrate how they can be used for another attack. The resources of the Kad network's peers and our attacks can be used to drive a *distributed denial of service attack* (DDoS) against any machine internal or external to the Kad network as follows: A node insertion attack is performed in order to occupy some popular keywords. Let μ be the machine (e.g., a server) to be attacked. We inform all requesters that μ contains the desired files. Consequently, all requests are directed to the attacked machine. Of course, the resulting load on μ is not larger than on the machine performing the node insertion. However, the advantage of this attack is that the attacking machine *remains hidden*; moreover, it is generally harder to counter a distributed DoS attack than a normal DoS attack as the requests originate from different (and valid) IP addresses. Also the Publish Attack can be used for the DDoS attack if we advertise wrong IP bindings of keywords. This has the additional advantage that the attack induces more load on the attacked machine than on the attacker, as the different Kad peers are directed to the attacked machine directly. Note that DDoS attacks using a p2p system such as Kad are particularly nasty as the peers store information about sources for a long period of time, implying that such an attack could last several days with steadily changing peers involuntarily performing the attack.

As all the described attacks can be performed easily and have a large impact, it is mandatory to derive and implement counteractive measures. In order to overcome the node insertion attack it must be guaranteed that choosing specific IDs is infeasible. A straightforward approach, which is often described in literature (and which is used, e.g., by the Azureus BitTorrent client), is to bind the ID directly to the peers' IP addresses, e.g., by hashing the IP address. However, there are several reasons why real-world p2p systems do not adhere to this simple rule. First, multiple peers may share the same IP address, for example, peers in a local area network behind a NAT router are typically addressed using the same public IP address. These peers would all have the same peer identifier. Second, IP addresses are often given out dynamically and the assignment of addresses may change. In case of an ID-IP binding, this implies that peers have to rebuild their routing tables when reconnecting to the network with a new IP. Additionally, all the credits gathered by uploading data would be lost irretrievably because the peer ID changed and hence the peer cannot be recognized by other peers anymore. It seems that some of these problems can be solved easily and the IP address can still be incorporated into the ID, e.g., by hashing the IP address and a randomly chosen bit string to solve the NAT problem, or by using a different, randomly chosen ID for the credit system, together with a public and private key pair to protect it against misuse.[3] Hashing the IP address and a user-generated bit string is preferable to including the port as this would require a static assignment of ports, and switching ports would also lead to a new ID. However, the crucial observation is that creating such a binding is not sufficient to avert the attack in general, as long as the ID includes a user-generated part. Assuming that a hash function such as SHA-1 is used, an attacker can try out millions of bit string in a short period of time in order to generate an ID that is closest to the targeted keyword even in a network containing more than a million peers. These observations indicate that some form of peer authentication is required, which is hard to achieve without the use of a centralized verification service. As part of the strength of the network is its completely decentralized structure, relying on servers does not seem to be an acceptable solution.

A simple heuristic to render the Kad network more resilient to publish and eclipse attacks is to limit the amount of information a peer accepts from the same IP address, i.e., a peer does not allow that its entire contact list is filled by peers using the same IP address. This is also a critical solution as several peers behind a NAT may indeed have the same public IP address. What is more, an attacker with several IP addresses at its disposal can circumvent this security measure.

A crucial observation is that many of the discussed vulnerabilities do not only pertain to the Kad network, such attacks can be launched against any fully decentralized system that does not incorporate strong verification mechanisms. We believe that in recent literature, some promising approaches have been proposed, especially the work on join-leave attacks [13] by Scheideler, who studies how to spread peers over a virtual ID space $[0, 1)$ in a robust way. In [4], Awerbuch and Scheideler proposed a robust distributed (round-robin) random number generator. Interestingly, while constructing a single random number is difficult, it turns out that a *set of random numbers* can be generated by a group of peers in a scalable manner that is resilient to a constant fraction of adversarial peers. Unlike the verifiable secret sharing algorithm described in [2], their solution cannot fail if the initiating peer does not behave correctly, and a peer cannot rerun the protocol sufficiently many times until an ID is generated that falls into a desired range. This is certainly a desirable property to overcome the node insertion attacks described in this paper. However, important questions remain open, for instance, how to handle concurrent rejoin operations, or how to cope with ongoing DoS attacks.

[3] In fact, Kad already uses public and private keys to authenticate peers whenever a new session starts.

5 Related Work

Peer-to-peer networks have become the most popular medium for bulk data dissemination, and a large fraction of today's Internet traffic is due to p2p file sharing.[4] The immense computational resources of p2p networks are also attractive to attackers, and there is already a large body of literature on the subject [5,20].[5] Reasons to attack a p2p system can be manifold: For example, a peer may seek to perform a more or less passive "rational attack" [12] to be able to benefit from the system without contributing any resources itself (see, e.g., the *BitThief* BitTorrent client [9]).

While such selfishness can threaten a peer-to-peer system, which essentially relies on the participants' contributions, there are more malicious attacks seeking to harm the system directly. An attacker may, for example, strive to partition the system or to eclipse individual nodes. The eclipse attack [14], as also described in this work, can be used by a set of malicious peers to position themselves around a given peer in the network such that the peer's contact list consists only of the colluding peers. In a *Sybil attack* [7], a single entity creates multiple entities of itself in order to gain control over a certain fraction of the system. Such an attack can undermine redundancy mechanisms and is hard to counter in a completely decentralized environment. Attackers may also exploit a peer-to-peer system to efficiently spread a *worm* [21]. Furthermore, the resources of a p2p system may also be used to attack *any* machine connected to the Internet regardless of whether it is part of the peer-to-peer network or not. A *denial of service attack* can be launched in various p2p systems, e.g., Gnutella [1], Overnet [11], and BitTorrent [6]. During this attack, information about the victim, i.e., the targeted machine in the attack, is spread in the system. The victim is falsely declared as an owner of popular content, causing other peers searching for this content to contact the victim repeatedly. In BitTorrent, tracker information can be faked which leads peers to believe that the victim is a tracker for the desired content [6]. In the Kad network, DoS attacks can be launched by means of a redirection attack where a queried peer, the attacker, will return a response containing the address of the victim [19]. As mentioned before, the attacks presented in this work can also be used to launch a DoS attack.

The work closest in spirit to ours is the study of *index poisoning attacks* in FastTrack and Overnet [8]. Their index poisoning attack is akin to our publish attack where bogus information is pushed aggressively to the nodes responsible for the desired keywords. However, while this attack is also quite successful, it is not as effective in the Kad network as it is in FastTrack and Overnet. We showed that a different, even simpler poisoning attack is feasible and more effective. Moreover, our study of attacks in the Kad network is not limited to content poisoning and index poisoning, but also considers the eclipse attack to prevent peers from accessing a specific file. It is also worth pointing out that, in comparison to Kad, it is generally easier to perform attacks on Overnet, as it, e.g., does not check whether the sender of a publish message provided its own IP address as the owner of the file, and no cryptography is used for authentication.

While we believe that there are methods to contain the potential damage caused by such attacks to a certain extent, it is known that some sort of logically centralized entity

[4] See http://www.cachelogic.com/research/

[5] See also http://www.prolexic.com/news/20070514-alert.php/

is required to thwart attacks such as the Sybil attack [7]. There is also some interesting theoretical work on how to identify and exclude large sets of colluding peers [3]. However, the described techniques cannot be used to counter our attacks as we require only a very small number of attackers close to a given ID, which is not sufficient to raise suspicion. For a more thorough discussion of possible countermeasures against attacks in p2p networks, the reader is referred to the corresponding literature (e.g., [5]).

Finally, the *Kad network* itself has been the subject of various works. Stutzbach et al. [17] describe implementation details of Kad in eMule, and [18] presents crawling results on the behavior of Kad peers. Steiner et al. [15] collected an extensive, interesting set of data about the Kad network by crawling the network during several weeks. For example, they found that different classes of participating peers exist inside the network. Finally, [16] initiated the study of Sybil attacks in Kad. The authors propose to tie the possibility of obtaining a Kad ID to the possession of a cell phone number. Their solution therefore requires a centralized entity.

6 Conclusion

Structured peer-to-peer systems are likely to gain importance in the near future. This is mainly due to the fact that structured p2p networks have many desirable properties whose usefulness goes far beyond efficient file sharing. Driven by these properties, the use of DHTs or similar structured networks has been proposed as the foundation of the "future Internet" in order to overcome the deficiencies of today's Internet.

This paper has provided evidence that the Kad network, which is currently the only widely deployed p2p network based on a DHT, can be attacked with a small amount of computing resources such that access to popular files is denied. It is clear that such attacks could significantly lower the throughput of the entire system as the sought-after files are no longer found, and that this imposed censorship would frustrate the users. Moreover, the possibility of leveraging the immense computational resources of the entire system to attack arbitrary machines constitutes a serious threat. We argue that the presented attacks can basically be launched in any peer-to-peer system that does not incorporate sound peer authentication mechanisms.

We have discussed different approaches to overcome these vulnerabilities. While there are both practical and theoretical schemes that seem to improve the robustness, more research is needed how to apply them optimally "in the wild".

References

1. Athanasopoulos, E., Anagnostakis, K.G., Markatos, E.P.: Misusing Unstructured P2P Systems to Perform DoS Attacks: The Network That Never Forgets. In: Zhou, J., Yung, M., Bao, F. (eds.) ACNS 2006. LNCS, vol. 3989, pp. 130–145. Springer, Heidelberg (2006)
2. Awerbuch, B., Scheideler, C.: Towards a Scalable and Robust DHT. In: Proc. SPAA (2006)
3. Awerbuch, B., Scheideler, C.: Towards Scalable and Robust Overlay Networks. In: Proc. 6th Int. Workshop on Peer-to-Peer Systems, IPTPS (2007)
4. Baruch, A., Christian, S.: Robust Random Number Generation for Peer-to-Peer Systems. Theor. Comput. Sci. 410(6-7), 453–466 (2009)

5. Castro, M., Druschel, P., Ganesh, A., Rowstron, A., Wallach, D.S.: Secure Routing for Structured Peer-to-Peer Overlay Networks. In: Proc. OSDI (2002)
6. El Defrawy, K., Gjoka, M., Markopoulou, A.: BotTorrent: Misusing BitTorrent to Launch DDoS Attacks. In: Proc. 3rd Workshop on Steps to Reducing Unwanted Traffic on the Internet, SRUTI (2007)
7. Douceur, J.R.: The sybil attack. In: Druschel, P., Kaashoek, M.F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, p. 251. Springer, Heidelberg (2002)
8. Liang, J., Naoumov, N., Ross, K.W.: The Index Poisoning Attack in P2P File Sharing Systems. In: Proc. INFOCOM (2006)
9. Locher, T., Moor, P., Schmid, S., Wattenhofer, R.: Free Riding in BitTorrent is Cheap. In: Proc. HotNets (2006)
10. Maymounkov, P., Mazières, D.: A Peer-to-Peer Information System Based on the XOR Metric. In: Druschel, P., Kaashoek, M.F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429. Springer, Heidelberg (2002)
11. Naoumov, N., Ross, K.: Exploiting P2P Systems for DDoS Attacks. In: Proc. 1st International Conference on Scalable Information Systems, INFOSCALE (2006)
12. Nielson, S.J., Crosby, S.A., Wallach, D.S.: A taxonomy of rational attacks. In: Castro, M., van Renesse, R. (eds.) IPTPS 2005. LNCS, vol. 3640, pp. 36–46. Springer, Heidelberg (2005)
13. Christian, S.: How to Spread Adversarial Nodes?: Rotate!. In: Proc. STOC (2005)
14. Singh, A., Ngan, T.-W.J., Druschel, P., Wallach, D.S.: Eclipse Attacks on Overlay Networks: Threats and Defenses. In: Proc. INFOCOM (2006)
15. Steiner, M., Biersack, E.W., Ennajjary, T.: Actively Monitoring Peers in the KAD. In: Proc. 6th Int. Workshop on Peer-to-Peer Systems, IPTPS (2007)
16. Steiner, M., En-Najjary, T., Biersack, E.W.: Exploiting KAD: Possible Uses and Misuses. SIGCOMM Comput. Commun. Rev. 37(5), 65–70 (2007)
17. Stutzbach, D., Rejaie, R.: Improving Lookup Performance over a Widely-Deployed DHT. In: Proc. INFOCOM (2006)
18. Stutzbach, D., Rejaie, R.: Understanding Churn in Peer-to-Peer Networks. In: Proc. 6th Internet Measurement Conference, IMC (2006)
19. Sun, X., Torres, R., Rao, S.: Preventing DDoS Attacks with P2P Systems through Robust Membership Management. Technical Report TR-ECE-07-13, Purdue University (2007)
20. Wallach, D.S.: A Survey of Peer-to-Peer Security Issues. In: International Symposium on Software Security (2002)
21. Zhou, L., Zhang, L., McSherry, F., Immorlica, N., Costa, M., Chien, S.: A first look at peer-to-peer worms: Threats and defenses. In: Castro, M., van Renesse, R. (eds.) IPTPS 2005. LNCS, vol. 3640, pp. 24–35. Springer, Heidelberg (2005)